Bioinformatics, 32(13), 2016, 1925–1932
doi: 10.1093/bioinformatics/btw064
Advance Access Publication Date: 2 March 2016
Original Paper

Genome analysis

Assembly scaffolding with PE-contaminated mate-pair libraries

Kristoffer Sahlin[1,*], Rayan Chikhi[2] and Lars Arvestad[3]

[1]Science for Life Laboratory, School of Computer Science and Communication, KTH Royal Institute of Technology, Solna, Sweden, [2]CNRS, CRIStAL, UMR 9189, Villeneuve D'ascq 59650, France and [3]Swedish e-Science Research Centre, Science for Life Laboratory, and Department of Numerical Analysis and Computer Science, Stockholm University, Stockholm, Sweden

*To whom correspondence should be addressed.
Associate Editor: John Hancock

Received on June 9, 2015; revised on January 29, 2016; accepted on February 1, 2016

Abstract

Motivation: Scaffolding is often an essential step in a genome assembly process, in which contigs are ordered and oriented using read pairs from a combination of paired-end libraries and longer-range mate-pair libraries. Although a simple idea, scaffolding is unfortunately hard to get right in practice. One source of problems is so-called PE-contamination in mate-pair libraries, in which a non-negligible fraction of the read pairs get the wrong orientation and a much smaller insert size than what is expected. This contamination has been discussed before, in relation to integrated scaffolders, but solutions rely on the orientation being observable, e.g. by finding the junction adapter sequence in the reads. This is not always possible, making orientation and insert size of a read pair stochastic. To our knowledge, there is neither previous work on modeling PE-contamination, nor a study on the effect PE-contamination has on scaffolding quality.

Results: We have addressed PE-contamination in an update to our scaffolder BESST. We formulate the problem as an integer linear program which is solved using an efficient heuristic. The new method shows significant improvement over both integrated and stand-alone scaffolders in our experiments. The impact of modeling PE-contamination is quantified by comparing with the previous BESST model. We also show how other scaffolders are vulnerable to PE-contaminated libraries, resulting in an increased number of misassemblies, more conservative scaffolding and inflated assembly sizes.

Availability and implementation: The model is implemented in BESST. Source code and usage instructions are found at https://github.com/ksahlin/BESST. BESST can also be downloaded using PyPI.

Contact: ksahlin@kth.se

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Genome assembly is still a challenging process, especially for large genomes, and scientists experiment with different combinations of data and tools to reduce errors, improve contiguity and avoid ambiguity. An important step in the assembly process is scaffolding, in which contigs are ordered, oriented and joined to form a larger scaffold unit. The input is a set of contigs and one or several genome mappings of paired short reads from either paired-end (PE) sequencing or mate-pair (MP) sequencing.

Evaluations (e.g. Hunt *et al.*, 2014) have shown that scaffolders make many mistakes, perhaps more than one might expect from what could appear a straightforward computational problem. The input to a scaffolder is however both large and noisy, and the data characteristics can vary a lot depending on the organism and

assembler. Although contiguity and errors are the most important metrics to evaluate a scaffolder by, we came to note that there are other artifacts from scaffolders not reported in these metrics. We observed that assemblies could increase in size by up to 106% after scaffolding and this mostly affects fragmented assemblies. We call this effect *assembly inflation*. A successful scaffolding will have some assembly inflation due to, e.g. unsequenced regions, but it should in general be very small.

The type of technology, PE or MP, and their main parameter, the insert size, determine how far apart the reads are distributed on the genome and thereby at what distances contigs can be connected into scaffolds. Whether it is PE or MP also determines if the reads are read towards (PE) or apart (MP) from each other. There are numerous complications with PE and MP reads. For example, the insert size is not perfectly controlled and larger insert size typically means a larger variance in the distribution of distances between the reads, making scaffolding harder (Sahlin *et al.*, 2014). A largely ignored problem (on the computational side) is so-called PE contamination of MP libraries, which is a consequence of the MP library preparation. During the process, an unknown fraction of fragments that do not contain the circularization junction are sequenced. These misreads behave like PE reads, with opposite read direction from MP (see Fig. 1) and effectively with a much smaller insert size (Illumina, 2012). Hence, PE contamination reads may confuse a scaffolder that assumes an MP library is clean from contamination, suggesting a different relative order of contigs.

When designing a scaffolder, one can take the stance that PE contamination is (1) an experimental issue which should be controlled in the wet-lab and (2) it can be treated as noise in the MP that will get filtered away in a scaffolding optimization procedure anyway. Regarding (1), we probably have to accept PE contamination as a largely unavoidable difficulty and, in that case, we argue that the PE should be explicitly modeled in MP datasets to reduce errors. Although a decent MP library will contain more true MP reads than PE, and hence overshadow PE contaminants in many cases, assumption (2) will not hold for fragmented assemblies as there are many short contigs close to each other making PE links dominate MP links. The ambition can also be set higher: perhaps we can start utilizing PE contamination as valuable short-range information rather than nuisance that needs to be filtered away?

One may also ask how sensitive the current generation of standalone scaffolders are to MP libraries with a high amount of PE contamination? If scaffolders have been designed for near ideal datasets, what can one expect in a more difficult situation?

PE-contamination has been discussed in relation to integrated scaffolders in end-to-end assemblers such as ALLPATHS-LG (Gnerre *et al.*, 2011) and MaSuRCA (Zimin *et al.*, 2013). These methods rely on the orientation being observable, e.g. by finding and removing the adapter sequence. In practice that is not always the case, as a fraction of read pairs will not contain the adapter

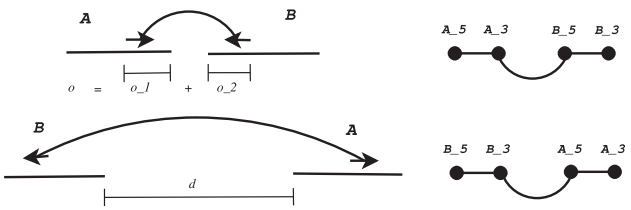

Fig. 1. Left: Two possible placements of contigs if we do not know the relative orientation of the read pair link. Right: The corresponding contig graphs where '5′ and '3′ denotes the 5′ and 3′ ends of contigs

(depending on shearing size) and orientation will not be observable. A method that does not rely on adapters being observable has the additional benefit of not putting a constraint on the fragment shearing size (in the re-fragmentation step of the circularized fragments in the library construction protocol). Smaller shearing size yields more 'identifiable' mate-pairs (adapter is present in the end of at least one read), but a higher contamination content of PE-fragments and chimeric reads (adapter present in the start or middle of a read). On the other hand, larger shearing size gives fewer identifiable mate-pairs but larger amount of true mate pairs as well as fewer PE-fragments and chimeric reads—due to the increased probability of sequencing full length reads on both sides of the adapter on a larger fragment. Scaffolding with a library with larger fragment shearing size also has the advantage that span coverage of the PE-contamination increases, giving better short range connections.

As this contamination has neither been modeled nor examined in terms of how it degrades scaffolding results, we decided to investigate this. We developed a scaffolder that models PE contamination reads in the scaffolding process on MP libraries. We show that modeling PE-contamination improves scaffolding. Our design utilizes an integer linear program (ILP) for the MP/PE classification and uses information from the interval structure of the contig graph to heuristically obtain a set of linear programs (LP) to solve. The heuristic method is very fast and finds the correct order of contigs in all cases given the model assumptions, as seen on our simulated data.

2 Methods

The presented work builds on the BESST scaffolder (Sahlin *et al.*, 2014), which iterates over PE and/or MP libraries in the order of their mean insert size. For each library, BESST scaffolds in two steps. In the first step, *safe linking*, large contigs are scaffolded using a greedy statistical scoring heuristic (a contig is large if it is unlikely that a read-pair can span over the contig). In a second step, *improved linking*, smaller contigs are placed within gaps of the larger contigs, or between larger contigs that were not linked in the first step (e.g. because their distance was larger than the insert size). The second step is done with a breadth-first search where the highest scoring paths (i.e. sets of contigs in the contig graph, see Sahlin *et al.*, 2014) are selected. The methods presented here are applied to the output of the *improved linking* step, that is, to paths of smaller contigs. Each path is a local subset of the contig graph, thus naturally dividing the contig graph into subregions in which the contigs needs to be oriented, ordered, and positioned accurately (deciding distance between contigs in a scaffold). If PE contamination is present, it can confuse the ordering as PE reads have the opposite orientation to MP reads. Figure 1 shows the two possible orderings for two contigs joined by a paired read link, notice however that the relative orientation between the contigs remains the same. We will here define the problem of ordering and positioning contigs with PE-contamination as an ILP. For m contigs linked with paired reads, we can have up to $m!$ possible orderings and it is not feasible to try them all. We therefore find a solution to this ILP by solving a reduced set of Linear Programs (LP) where each LP is induced by fixing the order of the contigs. With a fixed ordering of the contigs, solving the LPs will reduce to finding optimal gap sizes between contigs using the gap model by Sahlin *et al.* (2012).

2.1 Integer linear program formulation

The following paragraphs introduce the ILP formulation. We first consider a fixed contig ordering where the contigs are linked only by

MP links. We then add PE links to the formulation via an unknown variable representing the order of contigs.

2.1.1 Fixed ordering objective without contamination

Let a contig graph $\mathcal{G}$ be an undirected graph created by contigs and read pair links where each contig is represented by two vertices (for the 5′ and 3′ end respectively) and an intra-contig edge. Let an inter-contig edge be an edge in $\mathcal{G}$ that connects two vertices that come from different contigs if one or more read pairs suggest those two contigs are linked and the read pair(s) suggest the same orientation and distance of contigs (known as 'link-bundling', Huson *et al.*, 2002). The intra-contig edges are used in the implementation but does not contribute to the ILP problem and we will from now on only discuss inter-contig edges. Notice that a read pair can give rise to two different edges in $\mathcal{G}$, depending on if the read pair is assumed to be in PE or MP orientation, see Figure 1.

Assume that we have a connected subgraph of $\mathcal{G}$ induced by contigs $c_1, \ldots, c_m$ and a set E of inter-contig edges (not all contigs need to have edges between them), see supplemental Figure S1 for an example of a contig region. A linear order of m contigs has $m-1$ gaps $g_1, \ldots g_{m-1}$. Let us focus on the edge e between contigs c_i and c_j, defined by w_e links. Using only the links between c_i and c_j, assume that we have an estimate $\hat{d}_e$ of the distance between these two contigs, see Figure 1. Note that this distance is not to be confused with the gaps $g_1, \ldots g_{m-1}$, as c_i and c_j might not be adjacent in the given ordering. How to get the estimate of $\hat{d}_e$ accurately is described in (Sahlin *et al.*, 2012). From $\hat{d}_e$ and o_e (see Fig. 1), we naturally get the average insert size of the links that spans c_i and c_j as $\hat{\mu}_e = \bar{o}_e + \hat{d}_e$, where $\bar{o}_e$ is the average observation. That is, $\hat{\mu}_e$ is the average insert size suggested by the links between c_i and c_j. Notice that we define the insert size to include the read lengths (sometimes denoted the fragment length). In a given placement of the m contigs, c_i and c_j will be at distance $\sum_{k=i+1}^{j-1} |c_k| + \sum_{k=i}^{j-1} |g_k| + \bar{o}_e$, adding up the $j-i-1$ contigs and $j-i$ gaps between c_i and c_j. The objective is to minimize the discrepancy between the placement in the current ordering and $\hat{\mu}_e$, that is, to minimize

$$z_e := |\hat{\mu}_e - \left(\sum_{k=i+1}^{j-1} |c_k| + \sum_{k=i}^{j-1} |g_k| + \bar{o}_e \right)|.$$

Since each edge have w_e links, we would like to minimize

$$\min \sum_{e \in E} w_e z_e$$

taken over the possible orderings.

2.1.2 Fixed ordering objective with contamination

Let $\hat{\mu}_e^{\text{MP}}$ denote the estimated average insert size of links from edge e between two contigs given that the links are oriented as mate pairs in a given placement, and similarly let $\hat{\mu}_e^{\text{PE}}$ be the estimated average paired end insert size of edge e. Analogously, define the distance discrepancies z_e^{MP} and z_e^{PE} and let σ^{PE} and σ^{MP} be the standard deviations of the MP and PE distributions. Larger variance increases uncertainty in estimations $\hat{\mu}_e$, thus we weight the penalty relative to the standard deviations of the two distributions. Discrepancies from predicted MP distances are penalized with $\frac{\sigma^{\text{PE}}}{\sigma^{\text{PE}}+\sigma^{\text{MP}}}$ and discrepancies from predicted PE distances with $\frac{\sigma^{\text{MP}}}{\sigma^{\text{PE}}+\sigma^{\text{MP}}}$. As $\sigma^{\text{PE}} + \sigma^{\text{MP}}$ is constant, we omit this denominator in the objective. Note that higher uncertainty gives lower relative penalty. We have the objective function as

$$\min \sum_{e \in E} w_e z_e^{\text{MP}} \sigma^{\text{PE}} + w_e z_e^{\text{PE}} \sigma^{\text{MP}}.$$

As seen in the objective, edges with larger weights w will penalize a possible distance discrepancy more.

2.1.3 Fixed ordering constraints

Due to the MP library insert size, we do not allow gaps larger than $\mu^{\text{MP}} + k\sigma^{\text{MP}}$, with k empirically set to 2. Therefore, we have $m-1$ constraints

$$g_i \leq \mu^{\text{MP}} + k\sigma^{\text{MP}}.$$

Notice that even though a gap is an integer value, we choose to work with a relaxed problem to be able to apply LP.

2.1.4 Formulating an ILP for unknown orientation and ordering

The unknown in this problem is the orientation of contigs in a region. Let I_e^{MP} be the indicator function for edge e having MP orientation. The full ILP has the variables g, z and I^{MP} and is written as

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} [w_e z_e^{\text{MP}} \sigma^{\text{PE}} I_e^{\text{MP}} + w_e z_e^{\text{PE}} (1 - I_e^{\text{MP}}) \sigma^{\text{MP}}] \\ \text{subjectto} \quad & \sum_i g_i \leq \mu^{\text{MP}} + k\sigma^{\text{MP}} \\ & I_e^{\text{MP}} \in \{0, 1\}, e \in E \end{aligned}$$

We can use the minimum objective value to this ILP to evaluate contig orderings. For m contigs, there are $m!$ possible orderings, thus we need a way to efficiently choose a subset of orderings to run the ILP on.

For each assignment of $I_e^{\text{MP}}, e \in E$, the problem is a regular LP (remember the assumption $g \in R$) and we use common LP methods to express the LP on standard form, i.e. introducing help variables to remove absolute values and negative valued variables. This implies that we have $2(m-1) + |E|$ constraints in practice. The $|E|$ extra constraints comes from the fact that z_e is an absolute value and $2(m-1)$ constraints because the gap variables are allowed to be negative (letting $g_i = x_i - y_i$, for $x_i, y_i >= 0$). We solve the LP with the simplex method which we denote with $S(\cdot)$. S takes a path (translated into an LP instance) as argument and returns a real valued objective value.

2.2 Solving the ILP

BESST's *improved linking* step yields the ILP instances modeled in the previous section. A suggested path corresponds to a set of ordered contigs $c_1, c_2, \ldots, c_m$ where a consecutive pair c_i, c_{i+1} is connected by an edge with MP orientation, and c_1 and c_m are 'border' contigs (defined in Sahlin *et al.*, 2014)—their positions within the path are fixed. We call the edges connecting consecutive contigs c_i, $c_{i+1}, i \in [2, m-2]$ 'ambivalent'—the edges we infer PE or MP order over, see supplemental Figure S1 (non-consecutive contigs in the initial path may also be connected by edges). Let $p_0 = (c_1, c_2, \ldots, c_m)$ be a path found by BESST. Let E_0 be the set of indices i in p_0 where c_i, c_{i+1} are connected by an 'ambivalent' edge. Initially, a path of m contigs has exactly $m-3$ ambivalent edges, i.e. $|E_0| = m-3$, see supplemental Figure S1. Let $a(c, p)$ be the function that takes a contig c and an ordering p and returns the current position of c in the ordering p. For p_0, we create $m-2$ contig orderings $P_0 \doteq \{p_0^0, \ldots, p_0^{m-3}\}$ where p_0^0 in the identity permutation (initial ordering) and p_0^i is constructed by inserting c_i at position $a(c_{i-1}, p_0), \forall i \in E_0$. In this step, the permutation gives paths with contig orderings of the type $(c_1, \ldots c_i, c_{i-1}, \ldots c_m), \forall i \in [3, m-1]$. We choose the path with the

lowest objective for the next iteration as $p_1 = \text{argmin}_{p \in P_0} S(p)$. If $p_1 \neq p_0^0$ (i.e. there has been a change in ordering), we will remove the index i from E_0. This means that the edge between contig c_{i-1} and contig c_i is more likely to be a PE edge. That is, we have calculated the objective values of each candidate ordering formed by only permuting each consecutive pair of contigs, and we chose the one with the lowest objective. Notice that we 'switch' at most one MP link to a PE link by this procedure. We have now outlined the first iteration step of the algorithm.

A general iteration step j has three substeps. Substep 1: We form $m-j-2$ contig orderings $p_j^0, \ldots, p_j^{m-j-3}$ made from permutations, $\forall i \in E_j$, inserting segment $c_k, \ldots, c_i$ at position $a(c_{i-1}, p_{j-1})$, where $a(c_k, p_{j-1}) = a(c_{i-1}, p_{j-1}) + 1$, i.e. c_k comes immediately after c_{i-1} in p_{j-1}. Notice that the indices i always refer to the initial ordering in p_0^0 i.e. $(c_1, \ldots, c_m)$. In a general step j, c_{i-1} and c_i might no longer be adjacent (hence we needed to introduce the function $a(\cdot,\cdot)$). Substep 2: We calculate $S(p)$, $\forall p \in P_j$ and let

$$p_{j+1} = \underset{p \in P_j}{\text{argmin}} = S(p) \tag{1}$$

Substep 3: If $p_{j+1} = p_j^0$, we return the final path p_j^0. That is, no candidate orderings obtained a lower objective than the identity permutation. Otherwise, if $p_{j+1} = p_j^i$ for $i \neq 0$, we let $E_{j+1} = E_j \backslash i$. That is, we accepted the MP edge between c_i, c_{i+1} as being PE oriented. (4) Return to step (1). See Figure S1 for an illustration of the algorithm in practice. The algorithm stops either when $p_{j+1} = p_j^0$ or when the set of candidate edges are empty, which happens when $j = m-3$. This algorithm will at most compute $\frac{(m-3)(m-4)}{2}$ LPs. In practice, on the datasets we have scaffolded, m is almost always smaller than 15 and we have never observed m over 25.

Notice that changing relative orientation of contigs would yield pairs of orientation FF or RR, which are invalid according to our model, thus no such operation is allowed. A solution to each LP gives a set of real valued gaps g_i, $i \in [1, m-1]$ that are used to place the contigs accurately as information from several edges are used simultaneously for each gap.

The model is implemented in the workflow of BESST, and after applying BESST's scaffolding procedure (Sahlin *et al.*, 2014), we obtain ranked clusters of contigs (created by splitting the contig graph based on longer contigs) where the rank is based on how many supporting versus contradicting links a cluster has. Here, we start solving the ILPs, with the highest ranked cluster first. Paths containing contigs that are already included in higher ranked paths are discarded. As described, each ILP i is solved by solving a set of LPs. Our heuristic solution works well when the PE distribution has a short span. If the PE fragment length distribution is wide, and a significant number of PE-links would link non-neighboring contigs, more permutations would be needed. Thus, the permutation of adjacent contigs is efficient if most PE read pairs link adjacent contigs in the true ordering.

3 Results and discussion

We have compared our new implementation of BESST, called BESST-v2 below, with SSPACE (Boetzer *et al.*, 2011), OPERA (Gao *et al.*, 2011), SOPRA (Dayarian *et al.*, 2010), SCARPA (Donmez and Brudno, 2013) and SCAFFMATCH (Mandric and Zelikovsky, 2015). Version 1 of BESST (Sahlin *et al.*, 2014) is included for reference. We also include integrated scaffolder results provided by GAGE, labeled 'INTEGRATED'.

SSPACE, SCARPA, SOPRA and OPERA were run with mappings from Bowtie (Langmead *et al.*, 2009) as suggested by Hunt *et al.* (2014) when comparing the aligners BWA (Li and Durbin, 2010), Bowtie and Bowtie2 (Langmead and Salzberg, 2012). SCAFFMATCH is coupled with Bowtie2. BESST was run with BWA-MEM (Li, 2013). Full running instructions and details on resource usage are given in Supplementary data.

3.1 Datasets

We have included a simulated dataset, *sim*, which we developed the model on. A genome of size 500 000 bp was simulated and contigs of either 5000 ($P = 0.2$) or 500 bp ($P = 0.8$) were generated from this genome with no gap between contigs. The random generation of contig sizes ensure different complexities on the subproblems. The longest stretch of smaller contigs between border contigs consisted of 20 consecutive small contigs. Based on the reference genome, a MP library with 50× coverage of 2× 100 bp reads was simulated. The MP library had insert-size distribution $N(3000, 300)$ and 30% of the reads were PE contamination with insert size chosen from $N(400, 40)$.

Simulated data from the Assemblathon 1 study (Earl *et al.*, 2011) is denoted *assemblathon3k*. We assembled contigs with Minia (Chikhi and Rizk, 2013) from the paired-end reads provided by GAGE (Salzberg *et al.*, 2012). This gave approximately 74 000 contigs. The contigs were scaffolded with the 3 kbp MP library provided by Assemblathon 1. It contains mate pairs with insert-size distribution $N(3000, 300)$ and 20% PE contamination with an insert-size distribution of $N(500, 50)$.

Three cases, *Staphylococcus aureus*, *Rhodobacter sphaeroides* and human chromosome 14, were taken from the Genome Assembly Gold-standard Evaluation (GAGE) study (Salzberg *et al.*, 2012), a comprehensive evaluation of large-scale genome assembly algorithms. We denote them *staph*, *rhodo* and *hs14*, and GAGE provides 8, 9 and 9 contig assemblies, respectively. These assemblies were scaffolded with the original shortjump libraries provided by GAGE. By aligning these libraries to the reference genomes with BWA-MEM (Li, 2013) we detected a natural PE contamination in the rhodo and hs14 libraries at 41% and 33%, respectively, with mean insert size of 211 and 200 bp, respectively (see Supplemental Figs S2 and S3). The staph library had almost no PE contamination with the given alignments ($\ll$1%, see Fig. S1).

We created additional libraries for each reference genome to study the effect of increased levels of PE contamination and PE span coverage. The parameter c_{added} is introduced to indicate the percentage of added PE reads in a library. The original GAGE libraries have $c_{added} = 0$ (0% added contamination). The $c_{added} = 15$ libraries were formed by simulating PE reads from the genomes with distribution $N(300, 30)$ and adding them to the original read library. Similarly, the $c_{added} = 40$ libraries were formed by simulating PE reads from the genomes with distribution $N(400, 40)$ and adding them to the original library.

We used NxTrim (O'Connell *et al.*, 2015) and FastQC (http://www.bioinformatics.babraham.ac.uk/projects/fastqc) to find adapters within the GAGE datasets but a very small fraction of adapters were found (in 0–9% of the read pairs), hence MP and PE orientation could not be determined on the large majority of the reads—which inhibits the use of an adapter filtering tool to filter out PE-contamination. We therefore include additional evaluation of BESST-v2 scaffolding with all read pairs against BESST scaffolding with only MP-filtered read pairs on two datasets where a significant number of adapters can be found, see section S3 for detailed results and discussion.

3.2 Evaluation method

Assembly evaluation is known to be difficult (Earl *et al.*, 2011; Hunt *et al.*, 2014; Salzberg *et al.*, 2012), as there are many metrics to consider. Ultimately, the end result of an assembly should be as long error-free sequences as possible. However, two assemblies with the same lengths on error free sequences can still differ if one of the assemblers/scaffolders contains false joins, making the assembly look more contiguous. Thus, errors is another important metric. We therefore choose to look at the number of errors and the adjusted E-size (Salzberg *et al.*, 2012, the expected length of error free sequence in the assembly,) which we denote by E'.

Moreover, we need an informative way to present the quality increase/decrease of an assembly from contigs to scaffolds. Since the GAGE datasets contains assemblies of varying quality (with respect to contig errors and E') of each organism, we present the quality improvement between contigs and scaffolds. Let E'_c and E'_s denote the adjusted E-size of the original contig assembly and the scaffolded assembly. We report the increase in errors from original contigs to scaffolds as well as the ratio E'_s/E'_c. Tables S2–S18 in Supplementary data contains results for individual experiments and the summary tables presented here shows the average increase on each organism and library.

3.2.1 Assembly size inflation

As another aspect of scaffolding quality, we also show how total assembly size grows after scaffolding. We observed that some scaffolders increase the assembly size more than others and we wanted to find the cause. Since no scaffolder (in this comparison) puts a contig sequence in multiple places (e.g. repeats) or derive new sequence from the reads, inflation is due to added gaps (stretches of N's). We categorize inflation into three possible causes: (1) The genomic content between two contigs is not present in the assembly, thus an approximately correct number of N's is inserted. This is correct behaviour from a scaffolder and, to some extent, it improves the quality of the assembled genome. (2) The genomic content between two contigs a and c is present in another contig b in the assembly, but the scaffolder is unable to place b. The scaffolder creates (a correct number of) N's between a and c and leaves b isolated in the output file. This increases the assembly size, as there are two versions of a single location in the assembly, represented by b and N's respectively. It is not a misassembly, but it can confuse downstream analysis, and it negatively affects the assembly quality through size inflation. (3) Errors: a gap between contigs is inserted due to a false join, or the gap sequence significantly differs in length to the real gap length. This is seen as a misassembly that decreases the quality of the assembly. When we discuss assembly inflation we will refer to these three causes as (1), (2) or (3). We also report the ratio of scaffolded assembly size compared to initial contig size, labeled *inflation* in the tables, and find that some scaffolders are prone to (2).

3.2.2 Note on evaluation

Snakemake (Köster and Rahmann, 2012) was used to run our evaluation pipeline. QUAST (Gurevich *et al.*, 2013) was used for evaluating the scaffolders as it uses alignments to a reference to identify misassembly breakpoints. Support for computing E-size was added. QUAST classifies a misassembly as a breakpoint in the scaffold where the left and right flanking bases differs more than N base pairs from the reference sequence. We have set $N = 100$ to allow for reasonable variation in gap size estimations. Also, QUAST does not handle allele shifting of contigs on scaffolds from a diploid genome with a diploid reference. As the assemblathon3k dataset provides both copies of each chromosome, the evaluation was performed by giving only one of the copies of each chromosome (copy A) as reference to QUAST. Therefore, some of the errors occur due to the omitted reference copy. For example, the original assembly of minia is free from misassemblies if both copies is given as references. The scenario is however the same for all scaffolders so relative performance can still be compared.

3.3 Inflated assembly sizes

All scaffolders increase the assembly size to some extent but there is large variation among them. On the simulated dataset, all scaffolders except BESST-v2 and SCARPA inflate the assembly size by 18% or more even though all contigs have the potential to be linked (see Table 1). SCAFFMATCH inflates the assembly size the most (106.6%) and manual inspection reveals that this is due to a mix of cause (2) and (3). SCAFFMATCH uses an approach (maximum matching) that only permits one neighbour of every contig to be joined; a limitation on fragmented assemblies, making mate pairs link to several neighbors. SCAFFMATCH addresses this limitation by including a separate insertion-step that attempts to place remaining singleton contigs, but it does not perform well on this dataset. SSPACE also inflates the assembly size of the dataset sim to a large extent, but has only one error. The inflation is from (2), which is a result of SSPACE's heuristic; SSPACE extends scaffolds greedily by choosing (only) one neighboring contig with the most links. OPERA, SOPRA and BESST show vulnerability by making orientation mistakes (as in Fig. 1) due to creating many joins where the link is interpreted as a MP instead of a PE. SCARPA has low inflation on this dataset.

On the assemblathon3k dataset, see Table 2, BESST-v2 is able to increase the adjusted E-size with a factor of 22 while keeping the size of the assembly relatively constant. SCAFFMATCH, SSPACE, BESST and SCARPA inflate the assembly size significantly, but the large amount of errors and small E'_s/E'_c (for SSPACE and SCAFFMATCH) suggests that this is due to cause (3). As the PE oriented read has mean insert size of 500, this is the dataset where PE contamination has the highest span coverage and therefore most likely poses the biggest challenge.

On the GAGE datasets (Table 3) the inflation levels vary significantly among scaffolders. This phenomenon is most evident for SCAFFMATCH and SSPACE and is likely an artifact of their

Table 1. Results on the sim dataset. BESST-v2 finds the correct order, orientation and approximate positions of all contigs and joins the contigs into a single scaffold

Tool	Infl.	Scf-errors	E'_s/E'_c
BESST-v2	1.002	**0**	**130.9**
BESST	1.370	199	2.4
SSPACE	1.435	1	5.5
OPERA	1.406	255	1.0
SOPRA	1.179	73	2.0
SCARPA	1.015	9	5.4
SCAFFMATCH	2.066	95	2.5
	Size	Ctg-errors	E'_c
Contigs	496 500	0	3692.0

SSPACE gives only one error but places very few contigs and has an inflation of 43.5% from type (2). OPERA and BESST are the most sensitive to PE contamination with respect to misassemblies. 'Contigs' denotes the initial contig assembly metrics from which the relative inflation, increase in errors and corrected contiguity are computed.

Table 2. Assemblathon3k, the original Assemblathon 1 dataset with 3 kbp short jump library. BESST-v2 removes 83.6% of BESST's errors and has superior contiguity and number of misassemblies to the other scaffolders

Tool	Infl.	Scf-errors	E'_s/E'_c
BESST-v2	1.016	**2088**	**21.4**
BESST	1.152	15329	4.1
SSPACE	1.110	26410	1.0
OPERA	1.015	10161	1.7
SCARPA	1.070	15038	1.2
SCAFFMATCH	1.200	34124	1.0
	Size	Ctg-errors	E'_c
Contigs	120 105 427	7	5374.9

SOPRA is missing from the table because it did not meet the runtime constraint.

methodology. SCAFFMATCH inflates the assembly size the most and shows larger inflation as contamination increases (up to 19.2% and 28.7% on average on rhodo and hs14 respectively). SSPACE also shows high inflation rate on the GAGE assemblies already with the original libraries ($c_{added} = 0$), e.g. 6% on hs14. The contamination further worsens this behavior for SSPACE to 17% on average on rhodo and 9% on hs14 for $c_{added} = 40$. SCARPA and BESST are also affected by assembly inflation, but not to the same extent as SSPACE and SCAFFMATCH. Notably, SCAFFMATCH, SSPACE and SCARPA show an extreme inflation in assembly size on the two most fragmented assemblies on rhodo (ABySS and SGA) with inflation between 28 and 49% on ABYSS and 58 and 103% on SGA (Supplemental data, Table S7–S9), and a similar trend for the most fragmented assemblies on hs14 (Supplemental data, Table S11–S13). Such extreme inflation is clearly not correct as it almost doubles the assembly size, especially when the contig assembly size is already larger than the true genome size. We also argue that even subtle inflation increase on these datasets are artifacts as BESST-v2 in general has the fewest errors and highest increase in contiguity with only a small increase in assembly size. Inflation/deflation level vary among the integrated scaffolders. For example, SGA contig assemblies are generally significantly larger than the genome size and SGA's scaffolder removes a lot of sequence in the scaffolding step. In the hs14 assemblies, the integrated scaffolders in Velvet, Bambus2, MSR-CA and SOAPdenovo inflate assembly size the most and also introduce significantly more errors than the other scaffolders.

3.3.1 Summary inflation

In total, our results indicate that assembly inflation is more likely due to poor scaffolding (cause (3)) or the inability to place many contigs in a fragmented scaffold (cause (2)) rather than correctly added sequence gaps (1) for both stand alone and integrated scaffolders. This is supported by investigating the individual assemblies (Tables S2–S14). The more fragmented assemblies show significantly higher inflation and error rate than the higher quality ones. We suggest users of scaffolders to look at similar metrics after the scaffolding step is performed.

Notably, BESST-v2 reduces the assembly size slightly with increased PE contamination (Table 3). This is due to the fact that BESST-v2 can use the extra short range information to place smaller contigs, thus lowering inflation by reducing gapped sequence, i.e. gaps caused by (2).

3.4 Errors and contiguity

The sim and assemblathon3k datasets show how strongly PE contamination can affect scaffolding (Tables 1–2). There are extreme differences in inflation, errors and E'_s among the scaffolders on these two datasets. The differences in result between BESST and BESST-v2, as well as the number of errors of, e.g. OPERA, SOPRA and SCAFFMATCH, indicate that a large part of the links created in the scaffolding graph are from PE contamination. BESST-v2 corrects all errors on sim and the majority of errors on assemblathon3k, which indicates that it is an efficient method for finding the correct ordering.

The GAGE datasets (Table 3) further show that PE contamination affects scaffolding significantly, especially when assemblies are more fragmented, as is common for more complex genomes (see hs14, Table 3). OPERA and BESST seem to be the most vulnerable to contamination with significant increase in errors and lower E'_s across most assemblies as the contamination level increases. SCARPA is another scaffolder where introduced contamination changes the results drastically across single assemblies. However, one assembly (MSR-CA) is missing from SCARPA on hs14 due to our computational time constraint and the average is therefore somewhat distorted for hs14 with $c_{added} = 40$. The effect on individual assemblies is an increasing number of misassemblies (see Tables S12–S14).

SSPACE takes a more conservative approach as it only chooses one neighboring contig to extend the scaffold with, which is reflected with increased level of inflation. This results in a slightly lower E'_s for the $c_{added} = 40$ libraries. On rhodo (Table 3), however, there is a large increase in errors, where almost the entire increase is on the two fragmented assemblies ABySS and SGA. Due to the number of failed runs by SOPRA on hs14 (from the time constraint), no general conclusion about trend can be made with the averaged data presented here. However, as with the other scaffolders, SOPRA is the most affected by fragmented assemblies and greatly increases the number of errors on SGA and Velvet on hs14 when contamination is present (Table S12–S14). SCAFFMATCH shows fairly consistent number of misassemblies and E'_s across all datasets and with a relatively good E'_s. However, it always has significantly more misassemblies than other scaffolders.

With the GAGE libraries, BESST-v2 has the second fewest to fewest misassemblies and second highest to highest E'_s for almost all runs. The E'_s even increases on rhodo (Table 3) as contamination is introduced due to the extra read pair information. Although average number of misassemblies increases slightly for BESST-v2 with the contamination level in the GAGE runs, the difference is relatively small. In fact, the number of misassemblies at $c_{added} = 40$ made by BESST-v2 are competitive also at $c_{added} = 0$ with other scaffolders. The tradeoff between misassemblies and increase in E'_s gives BESST-v2 favorable results to the other scaffolders. Table 3 for hs14 with $c_{added} = 0$ shows that BESST-v2 lowers the misassemblies with almost 40%, with the most significant decrease on ABySS, ABySS2, SGA and Velvet. This suggests that a large part of the edges formed in a fragmented contig graph comes from PE contamination. Notably, BESST-v2 also improves scaffolding compared with integrated scaffolders across almost all datasets (see Tables S4, S7 and S12). On hs14, BESST-v2 has both fewer errors and higher corrected contiguity for all assemblies except for ABySS and ABySS2, where contiguity is 3-fold higher but at the cost of more errors.

In conclusion, stand-alone scaffolders introduce a large number of misassemblies on the fragmented assemblies with contamination present (e.g. ABySS and SGA in rhodo, Table S7–S9, or ABySS, SGA and Velvet in hs14, Table S12–S14). By comparing the two versions of BESST, we see that BESST-v2 corrects most of these

Table 3. GAGE contig assemblies for Staph, rhodo and hs14 scaffolded with GAGE's shortjump MP-library, $c_{added} = 0$, with an added 15% PE contamination reads, $N(300, 30)$, $c_{added} = 15$, and with an added 40% PE contamination reads, $N(400, 40)$, $c_{added} = 40$

staph	$c_{added} = 0$			$c_{added} = 15$			$c_{added} = 40$		
Tool	Infl.	Scf-errors	E'_s/E'_c	Infl.	Scf-errors	E'_s/E'_c	Infl.	Scf-errors	E'_s/E'_c
BESST-v2	1.017	16.5	11.3	1.013	37.0	**8.3**	1.015	37.9	**4.9**
BESST	1.017	16.5	11.3	1.039	97.6	2.7	1.04	98.8	2.3
OPERA	1.011	**14.5**	**11.5**	1.015	119.0	1.0	1.019	139.9	1.0
SCARPA	1.007	18.5	1.5	1.026	27.3	2.0	1.026	34.1	2.2
SCAFFMATCH	1.049	89.6	2.7	1.073	75.3	2.8	1.089	89.8	2.8
SOPRA	1.009	23.4	4.7	1.017	**25.5**	2.5	1.016	**19.5**	2.2
SSPACE	1.017	23.0	3.1	1.031	28.4	3.0	1.041	28.1	2.9
INTEGRATED	0.994	23.4	4.3						
	Size	Ctg-errors	E'_c						
Contigs	3 180 534	21.8	71 410.5						
rhodo	$c_{added} = 0$			$c_{added} = 15$			$c_{\text{added}} = 40$		
Tool	Infl.	Scf-errors	E'_s/E'_c	Infl.	Scf-errors	E'_s/E'_c	Infl.	Scf-errors	E'_s/E'_c
BESST-v2	1.031	36.7	**8.2**	1.024	**55.2**	**7.6**	1.018	**36.4**	**8.3**
BESST	1.04	47.8	7.9	1.062	170.1	6.1	1.062	182.6	6.1
OPERA	1.015	146.4	2.3	1.012	206.0	1.0	1.018	439.3	1.0
SCARPA	1.027	45.2	1.5	1.08	67.6	1.0	1.116	114.6	1.1
SCAFFMATCH	1.11	186.8	4.8	1.135	182.0	4.8	1.192	197.3	4.6
SOPRA	1.018	30.7	1.6	1.071	88.9	1.5	1.115	81.1	1.4
SSPACE	1.017	**25.2**	1.5	1.06	76.0	1.4	1.167	155.3	1.4
INTEGRATED	1.033	46.4	5.4						
	Size	Ctg-errors	E'_c						
Contigs	4 640 197	63.0	24 662.2						
hs14	$c_{added} = 0$			$c_{added} = 15$			$c_{added} = 40$		
Tool	Infl.	Scf-errors	E'_s/E'_c	Infl.	Scf-errors	E'_s/E'_c	Infl.	Scf-errors	E'_s/E'_c
BESST-v2	1.03	**2380.3**	**3.8**	1.026	**2808.1**	**3.5**	1.019	2442.8	**3.2**
BESST	1.046	4292.4	3.0	1.059	6304.2	2.7	1.051	6266.0	2.4
OPERA	1.024	4854.3	1.5	1.007	5587.4	1.1	1.021	7526.9	1.0
SCARPA	1.036	2990.7	1.6	1.046	3499.6	1.9	1.046	3490.3	1.8
SCAFFMATCH	1.128	9345.3	2.3	1.132	9321.7	2.3	1.287	9687.7	2.4
SOPRA	1.044	5116.1	2.4	1.045	4382.7	2.2	1.011	**1109.8**	1.7
SSPACE	1.063	3940.3	2.6	1.065	3913.9	2.6	1.093	3650.8	2.5
INTEGRATED	1.079	7238.2	2.5						
	Size	Ctg-errors	E'_c						
Contigs	985 16181	2109.1	14 087.4						

The numbers are averaged over each assembly. Full tables are found in Supplementary data. 'Contigs' denotes the initial contig assembly metrics from which the relative. Boldfaced numbers indicates fewest scaffolding errors and highest ratio of increase in corrected E-size between contigs and scaffolds.

misassemblies (see e.g. ABySS and SGA assemblies on rhodo, Table S7–S9 and hs14, Table S12–S14, in Supplementary data). Modeling PE contamination in the scaffolding step is important for larger and more complex genomes where assemblies tends to be fragmented, thus the proportion of PE links increases. This is supported by looking at the number of errors that is corrected in the hs14 assemblies on real data; with original GAGE libraries BESST-v2 reduce 54% of BESST's errors when scaffolding the SGA assembly and 53% on the ABySS assembly (See Supplemental Table S13). Finally, BESST-v2 generally gives preferable results over integrated scaffolders on GAGE data.

3.4.1 Runtime and alignments

We also measured runtime and peak memory of the tools, see Supplementary Tables S36–S47. Resource usage is not the main objective in this study and has been studied in other work (Hunt *et al.*, 2014; Sahlin *et al.*, 2014). However, we note that BESST, BESST-v2, SSPACE, OPERA and SCAFFMATCH (with the greedy approach) all have runtimes that should be practical on most genomes. There is no big difference in speed and memory demand between BESST-v2 and BESST.

4 Conclusions

We have designed a scaffolder that can identify and make use of read pairs with PE orientation in a MP library, so called PE contamination. The scaffolder (BESST-v2) accurately infers scaffolds, even with high levels of contamination, and we showed that other scaffolders are vulnerable to PE-contaminated libraries. Our results indicate that, when modeled, PE contamination helps scaffolding, serving as short-range linking information which complements long-ranging mate-pair reads. This combination of reads helps placing

small contigs in fragmented assemblies. We also showed that inflated assembly sizes after scaffolding are more often a result of the inability of scaffolders to place all contigs in a scaffold or erroneous gaps, rather than correctly inserted missing sequence from the contig assembly.

Funding

This work was in part funded by the Swedish Research Council (grant 2010-4634). The computations were performed on resources provided by the Swedish National Infrastructure for Computing through Uppsala Multidisciplinary Center for Advanced Computational Science (UPPMAX) under project b2013169.

Conflict of Interest: none declared.

References

Boetzer,M. *et al.* (2011) Scaffolding pre-assembled contigs using SSPACE. *Bioinformatics*, **27**, 578–579.

Chikhi,R. and Rizk,G. (2013) Space-efficient and exact de bruijn graph representation based on a bloom filter. *Algorithms Mol. Biol.*, **8**, 22.

Dayarian,A. *et al.* (2010) SOPRA: Scaffolding algorithm for paired reads via statistical optimization. *BMC Bioinf.*, **11**, 345.

Donmez,N. and Brudno,M. (2013) SCARPA: scaffolding reads with practical algorithms. *Bioinformatics*, **29**, 428–434.

Earl,D. *et al.* (2011) Assemblathon 1: a competitive assessment of de novo short read assembly methods. *Genome Res.*, **21**, 2224–2241.

Gao,S. *et al.* (2011) Opera: reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. *J. Comput. Biol.*, **18**, 1681–1691.

Gnerre,S. *et al.* (2011) High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proc. Natl. Acad. Sci. USA.*, **108**, 1513–1518.

Gurevich,A. *et al.* (2013) QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, **29**, 1072–1075.

Hunt,M. *et al.* (2014) A comprehensive evaluation of assembly scaffolding tools. *Genome Biol.*, **15**, R42.

Huson,D.H. *et al.* (2002) The greedy path-merging algorithm for contig scaffolding. *J. ACM*, **49**, 603–615.

Illumina (2012). Data processing of Nextera mate pair reads on Illumina sequencing platforms. Technical note.

Köster,J. and Rahmann,S. (2012) Snakemake – a scalable bioinformatics workflow engine. *Bioinformatics*, **28**, 2520–2522.

Langmead,B. and Salzberg,S.L. (2012) Fast gapped-read alignment with bowtie 2. *Nat. Methods*, **9**, 357–359.

Langmead,B. *et al.* (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.*, **10**, R25.

Li,H. (2013). Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv:1303.3997*.

Li,H. and Durbin,R. (2010) Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics*, **26**, 589–595.

Mandric,I. and Zelikovsky,A. (2015) Scaffmatch: Scaffolding algorithm based on maximum weight matching. *Bioinformatics*, **31**, 2632–2638.

O'Connell,J. *et al.* (2015) Nxtrim: optimized trimming of illumina mate pair reads. *Bioinformatics*, **31**, 2035–2037.

Sahlin,K. *et al.* (2012) Improved gap size estimation for scaffolding algorithms. *Bioinformatics*, **28**, 2215–2222.

Sahlin,K. *et al.* (2014) BESST – efficient scaffolding of large fragmented assemblies. *BMC Bioinf.*, **15**, 281.

Salzberg,S. *et al.* (2012) GAGE: A critical evaluation of genome assemblies and assembly algorithms. *Genome Res.*, **22**, 557–567.

Zimin,A.V. *et al.* (2013) The MaSuRCA genome assembler. *Bioinformatics*, **29**, 2669–2677.